An Improved Safety Solver in Go Using Partial Regions

Xiaozhen Niu and Martin Müller

Department of Computing Science, University of Alberta,
Edmonton, AB, Canada, T6G 2E8
{xiaozhen,mmueller}@cs.ualberta.ca

Abstract. Previous safety-of-territory solvers for the game of Go have worked on whole regions surrounded by stones of one color. Their applicability is limited to small to medium-size regions. We describe a new technique that is able to prove that parts of large regions are safe. By using pairs of dividing points, even huge regions can be divided into smaller partial regions that can be proven much easier and faster. Our experimental results show that the new technique significantly improves the performance of our previous state of the art safety-of-territory solver. Especially in earlier game phases, the solver utilizing the new technique outperforms the previous solver by a large margin.

1 Introduction

Evaluating the safety of territories is one of the most important components of a Go program. The previous work in [4,5,6] introduces several search-based safety-of-territory solvers that can determine the correct safety status of a given region. One weakness of the previous solvers is that their applicability is limited to small to medium-size regions. The reason is that the search space grows exponentially with region size. In real games, most often a board contains very large regions. When more and more stones are played, these large regions are gradually divided into smaller regions. Therefore the safety-of-territory solver can only be applied in the late stage of a game.

This paper introduces a new technique that can prove the safety of parts of large regions. By applying a miai strategy to pairs of dividing points, a large region can be divided conceptually into smaller partial regions. Separate safety searches can then be performed on each of these smaller partial regions. The experimental results show that the partial proving module improves the performance of a state of the art safety-of-territory solver. Even early in games, when there are only large regions on the board, the current system can prove the safety of many partial regions and their surrounding blocks.

The structure of this paper is as follows. Section 2 briefly discusses related work. Section 3 explains details of the partial proving module. Section 4 discusses experimental results, and the final section provides conclusions and further research directions.

H.J. van den Herik et al. (Eds.): CG 2008, LNCS 5131, pp. 102–112, 2008.

2 Related Work

There are many successful approaches for safety recognition proposed in the literature. The classical algorithm due to Benson statically recognizes *unconditionally alive* blocks and regions on board [1]. A number of papers address the question of eye shape of a region surrounded by a single block. Vilà and Cazenave's static classification rules evaluate many such regions of size up to 7 points as safe [7]. Dyer's eye shape library contains eye shapes up to size 7 [2]. Wolf and Pratola extend the analysis to size 11 regions, and compute many interesting properties of such regions such as ko status [9].

Müller identifies regions that are safe by alternating play [3]. The work introduces the notion of miaipairs for statically proving the safety of regions that can make two eyes in two independent ways. Van der Werf presents a learning system for high-accuracy heuristic scoring of final positions in the game of Go [8].

The current work extends the safety solvers described in [4,5]. *SAFETY SOLVER 1.0* is the previously best solver for evaluating the safety of completely enclosed regions. It can solve regions with size up to 18 empty points in reasonable time. *SAFETY SOLVER 2.0*, described in [5], can handle open boundary regions. Its board partitioning is based on open boundary zones. Its size limitation is similar. The current paper focuses on recognizing safe partial regions inside large regions with no size limitation.

3 Using Partial Regions for Safety Recognition

This section describes the four major processing steps and the related algorithms that are implemented to prove partial regions safe.

3.1 Find Dividing Miaipairs

To prove parts of a region R as safe, it must first be divided into reasonable chunks. A simple miai strategy is utilized for this purpose. A *miaipair* [3] is a pair of two empty points inside R, such that the defender playing at either of these two points would split R into two subregions. A defender miai strategy applied to these two points forces the defender to occupy at least one of these points: whenever the attacker plays one point and the other one is still empty, the attacker is forced to reply there.

This paper focuses on miaipairs containing two adjacent points which are also adjacent to defender boundary stones. Let $L(R)$ be the set of all splitting points inside R which are liberties of boundary blocks of the region. In the example on the left of Fig. 1, $L(R) = \{c1, e1, f1, j1\}$. The only miaipair is $\{e1, f1\}$. The black region on the right of Fig. 1 contains two overlapping miaipairs $\{d1, e1\}$ and $\{e1, f1\}$.

3.2 Dividing a Single Region Using One Miaipair

The simplest approach uses a single miaipair to divide a region R. Figure 2 shows a large black region R with miaipair $P = \{o1, p1\}$. By following the miai

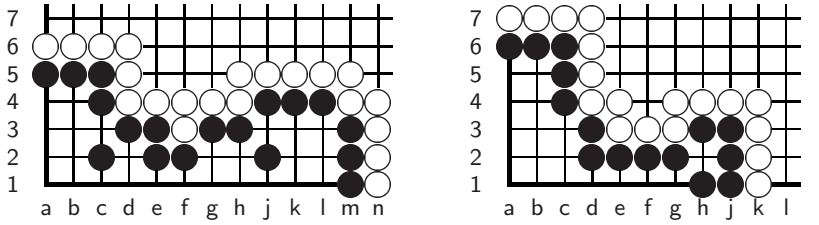

Fig. 1. Examples of miaipairs inside black regions

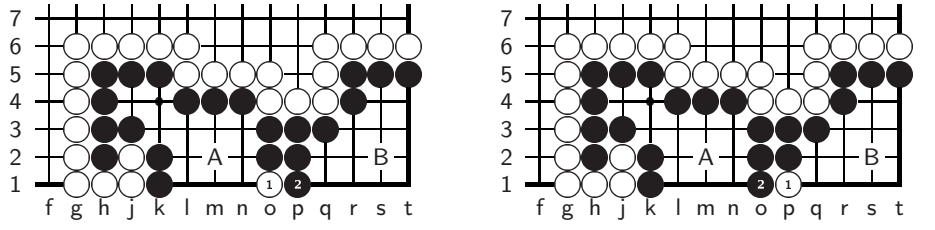

Fig. 2. Single region dividing by using miaipair $\{o1, p1\}$

strategy, Black can divide R into two subregions, A on the left and B on the right.

Assume a region R is divided into two open boundary subregions A and B by a miaipair $M = \{p_A, p_B\}$, such that p_A is adjacent to A and p_B is adjacent to B. Local safety searches are performed for $A \cup M$ and $B \cup M$. The safety search in an open boundary region is similar to the one described in [6], but constrained by the miai strategy outlined above as follows (shown for $A \cup M$).

1. The attacker can select any legal move in $A \cup M$, as long as both points in M are empty.
2. The defender checks whether a forced miai reply exists. If the attacker just occupied either p_A or p_B and the other miai point is still open, the defender immediately replies there.
3. Otherwise, the defender can choose any legal move in $A \cup \{p_A\}$ (but not p_B).

For example, when searching A in Fig. 2, White as the attacker can select any move in A as well as both moves from the miaipair $\{o1, p1\}$. Black can choose the same moves except $p1$. If White plays first at $o1$ or $p1$, Black must take the other. However if Black plays $o1$ first, the miai strategy is fulfilled and conditions need not be checked in the future. The move $p1$ is also removed from White's options.

The basic algorithm to prove that a dividable single region R is safe by using a miaipair-constrained safety solver is shown below. The algorithm takes another parameter S, the set of points (possibly including boundary blocks) previously shown to be safe by using other regions.

1. Use miaipair M to divide region R into two open boundary subregions A and B.
2. Run solver for A and compute new set of safe points: $newS$=$solve(A, \{M\}, S)$.
3. A was proven safe iff $newS \neq S$.
   (a) If $newS \neq S$, then use $newS$ to try to prove subregion B: $S = solve(B, \{M\}, newS)$.
   (b) If $newS = S$, then run solver on B: $newS = solve(B, \{M\}, S)$. If B is safe, then try to use the newly proven boundary blocks of B to prove A again: $S = solve(A, \{M\}, newS)$.

The result can be summarized as follows.

- If both A and B were proven safe, then R and all its boundary blocks are proven safe.
- Otherwise, if exactly one subregion is proven safe, then that region, its surrounding blocks, and the closer miai point are marked as safe. In the example, if only A were proven safe, then A, its boundary blocks, and $p_A = o1$, (a total of 30 points) would be marked as safe.
- If both local searches fail, nothing is proven safe.

For the example in Fig. 2, both sides and therefore the whole region can be proven safe by our system. In Fig. 3, the original large black region (size: 31) is divided into two subregions by miaipair $\{o1, p1\}$. Only the subregion in the right corner can be proven as safe. Its territory is marked by S and the safe boundary block is marked by triangles.

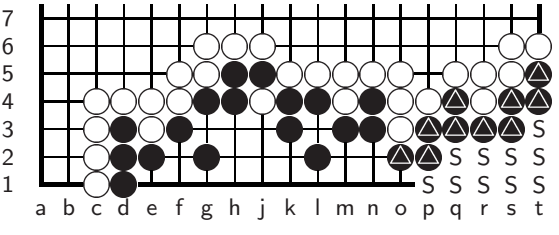

Fig. 3. Part of region is proven safe

3.3 Dividing a Single Region by Multiple Miaipairs

The basic method of Subsection 3.2 is restricted to single miaipairs within a region. Regions with multiple miaipairs can potentially be subdivided in many different ways. The easy case is *independent miaipairs*, where no two pairs are adjacent or overlap.

Figure 4 shows an example. The black region R of size 40 contains two independent miaipairs $P_1 = \{f1, g1\}$ and $P_2 = \{o1, p1\}$ that divide R into three partial regions, A to the left of P_1, B between P_1 and P_2, and C to the right of P_2. Since B is bounded by two miaipairs, both will be passed to the search

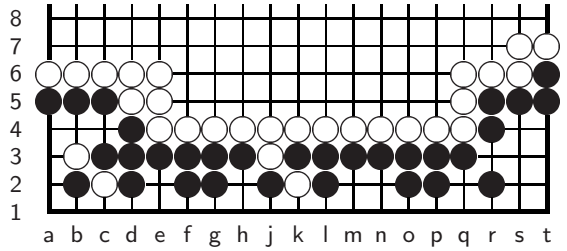

Fig. 4. First case of using multiple miaipairs together to divide a large region

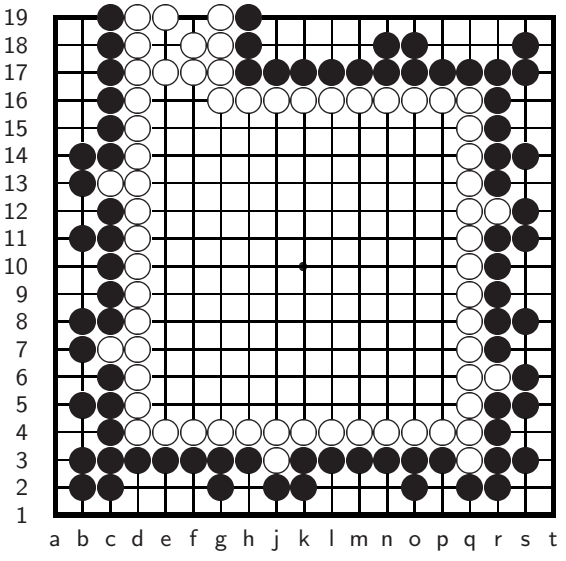

Fig. 5. Using 10 miaipairs to prove safety of a large black region

$solve(B, \{P_1, P_2\}, S)$. In general, a partial region bounded by n independent miaipairs is divided into $n+1$ partial regions that can be searched separately. Every time a partial region is proven safe, S is updated to include the region and its safe boundary. Then, subregion searches continue until no further updates can be made.

Figure 5 shows an extreme case from our test set, a huge black region of size 174. It contains 10 independent miaipairs. No previous search method can prove its safety. Using only a single miaipair at a time, just two small partial regions at the ends can be proven safe. The complete method proves the safety of the whole region.

If miaipairs are adjacent to each other or overlap, they cannot all be used. For example in the right of Fig. 1, the two miaipairs $P_1 = \{d1, e1\}$ and $P_2 = \{e1, f1\}$, cannot divide the region into three subregions. In this case the current implementation first computes clusters of miaipairs which are adjacent or overlapping,

then selects a single pair from each cluster. The selection is originally biased towards miaipairs that minimize the size of the largest subregion, but can be modified by backtracking if subproblems fail. In this example, miaipair P_1 is chosen first to find the largest possible safe area. If the whole region cannot be proven as safe by using P_1, then other miaipairs in this cluster will be tried.

3.4 Dividing a Merged Region

A set of strongly or weakly related regions can be merged into a large region, as described in [5]. After dividing a region, the resulting subregions may need to be merged with their respective strongly or weakly related regions. Figure 6 shows an example from the test set.

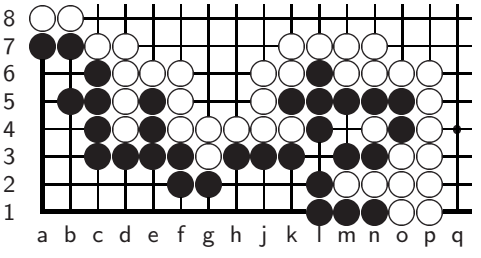

Fig. 6. Dividing a merged region

The original algorithm from [5] merges the regions r_1 at $a1$, r_2 at $l3$, and r_3 at $m4$ into a single region. r_1 contains the only miaipair $\{f1, g1\}$. The current algorithm first splits r_1 into subregions A on the left and B on the right. At this point related regions need to be merged. In the example, A has no related regions and can be tested on its own, but since B is strongly related to subregions r_2 and r_3, the (sub)regions B, r_1, and r_2 are merged into a new partial region for testing.

4 Experimental Results

SAFETY SOLVER 3.0 enhances the previous *SAFETY SOLVER 1.0* and *SAFETY SOLVER 2.0* with the new partial-region proving module.

There are two test sets for experiments to test the performance of *SAFETY SOLVER 3.0*. The first set contains 21 test positions. Each position contains either a large single region or a large merged region [5]. 17 of these test positions are taken from master games. The remaining 4 positions were created by the authors to test extreme cases with many miaipairs. The second test set is the collection of 67 master games introduced in [6]. The test sets are available at: `http://games.cs.ualberta.ca/go/partial`.

All experiments were performed on a Pentium IV/1.7GHz machine with 1 Gb memory. The following abbreviations for the solvers and enhancements are used in the text.

BENSON. Benson's algorithm, as in [3].
STATIC. Static safety-of-territory solver from [6].
SOLVER 1.0. Search-based safety-of-territory solver as described in [5]. It uses regions for board partitioning.
SOLVER 1.0 + P. Solver 1.0 + partial-region proving module.
SOLVER 2.0. Open boundary safety-of-territory solver as described in [6].
SOLVER 3.0. Solver 2.0 + partial-region proving module, the full solver.

4.1 Experiment One: Partial Region Solving

The purpose of this experiment is to test the performance improvements of the partial-region proving module in *SOLVER 1.0*. Since *SOLVER 2.0* uses heuristically computed open boundary zones for board partitioning, many positions in this experiment cannot be recognized. Therefore *SOLVER 2.0* is not compared in this subsection. For all 21 positions, the time limit is set to 200 seconds per search.

The only test position not solved by *SOLVER 1.0 + P* is shown in Fig. 7. The white region (size: 25) can be nicely divided into two small subregions A (size: 11) and B (size: 12) by using miaipair $\{k19, l19\}$. However, neither subregion can be proven safe due to the conservative assumption that no external liberties of boundary blocks may be used to establish safety. For example, when searching subregion A on the left, after move sequence $(B : k19, W : l19)$ White's boundary block at $k18$ is considered to be in atari by the solver because the external liberties at $m18$ and $m19$ may not be used. The situation for proving subregion B is analogous.

For the remaining 20 positions, *SOLVER 1.0 + P* finds at least some safe partial regions. Most of these 20 positions have size larger than 18 points. *SOLVER 1.0* can only prove 4 of them safe within 200 seconds. For a further analysis of the performance improvements, we divide these 20 positions into three groups.

Group 1 contains the 4 test positions that can be proven safe by both *SOLVER 1.0* and *SOLVER 1.0 + P*. Table 1 compares the solution time and number of expanded nodes for both solvers. In all 4 positions, the partial-proving module greatly improves the solver's performance. For example, Fig. 8 shows Position 21 at the top left corner and Position 11 at the bottom right corner. *SOLVER 1.0 + P* is over 61 times faster than *SOLVER 1.0*. However when solving Position 21 (size: 18), *SOLVER 1.0 + P* is only 3.3 times faster. The partial-region proving module first finds the most evenly dividing miaipair $\{r1, s1\}$, then performs

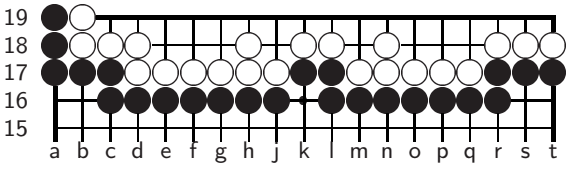

Fig. 7. The only position in set 1 that can not be proven safe

Table 1. Comparison of performance improvements

Position		*SOLVER 1.0*		*SOLVER 1.0 + P*	
Name	Size	Time (Seconds)	Nodes Expanded	Time (Seconds)	Nodes Expanded
No.2	15	8.5	25,993	2.10	1,785
No.8	16	25.55	70,133	1.64	1,752
No.11	18	120.13	236,786	35.7	45,123
No.21	18	156.57	232,332	2.53	3,506

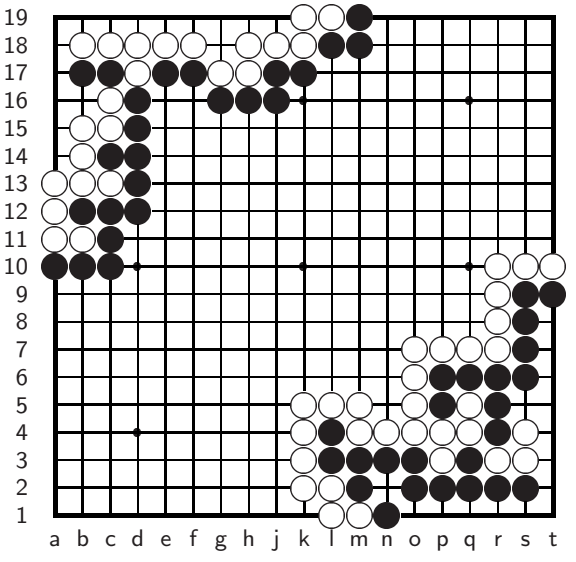

Fig. 8. Examples from group 1

safety searches to prove the whole region safe in 35.7 seconds. Our conclusion is that by using the miaipair $\{b19, c19\}$the division in Position 21 is quite even, each partial region has a similar small size. Therefore each local search is very fast. In contrast, the division in Position 11 is not that even. When using the most evenly dividing miaipair $\{p1, q1\}$, the left and right partial regions have the sizes of 4 and 12. Therefore the local search in the right partial region still requires longer time.

Group 2 contains the 6 test positions that can only be proven partially safe by *SOLVER 1.0 + P*. The top of Fig. 9 shows a real game position from this group. The program cannot prove the whole black region (size: 50) safe. However, by using the miaipair $\{p19, q19\}$ it proves that the partial region S at the top right corner and its boundary blocks (marked by triangles) are safe in 0.47 seconds.

Group 3 contains 10 test positions with either a large single region or a large merged region. *SOLVER 1.0 + P* can prove the whole region safe for every position. The bottom of Fig. 9 shows a real game position from this group. The black merged region contains two subregions r_1 at $b2$ and r_2 at $e3$. The size

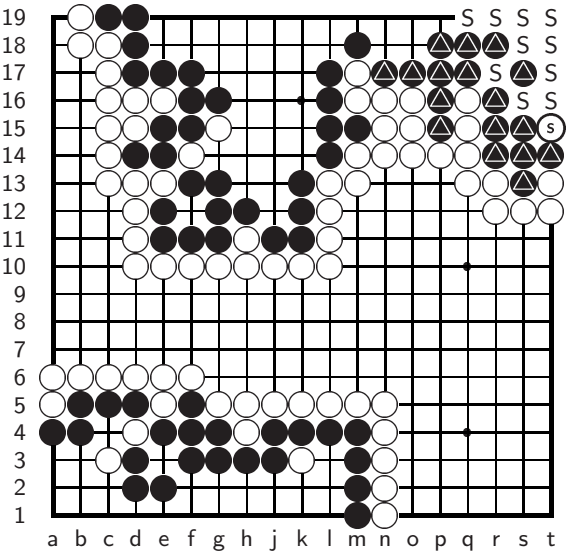

Fig. 9. Examples from group 2 and 3

of the merged region is 28. By using the miaipair $\{d1, e1\}$, *SOLVER 1.0 + P* proves it safe in 72 seconds. A second example from this group is the position shown in Fig. 5, which contains a huge region (size: 174) with multiple miaipairs. *SOLVER 1.0 + P* proves this extreme case safe in 63 seconds.

4.2 Experiment Two: Comparison of Solvers

This experiment compares the performance of solvers *BENSON*, *STATIC*, *SOLVER 1.0*, *SOLVER 2.0*, *SOLVER 1.0 + P* and *SOLVER 3.0* on 67 completed games. The time limit is set to 20 seconds per search. As in [6], each solver computes the proven safe points starting from the end of the game, then backwards every 25 moves. Table 2 shows the total number of proven safe points for all six solvers.

Table 2. Comparison of solvers on 67 games

Game Phases	End-100	End-75	End-50	End-25	End
BENSON	19	63	257	600	2,571
STATIC	106	242	587	1,715	5,584
SOLVER 1.0	234	462	1,138	3,189	10,285
SOLVER 2.0	594	838	1,651	3,653	10,815
SOLVER 1.0 + P	292	540	1,704	3,765	11,299
SOLVER 3.0	606	884	2,179	4,227	11,725

SOLVER 3.0 can prove the most points safe in all game phases. Interestingly, in earlier game phases such as *End - 100* and *End - 75*, the open boundary *SOLVER 2.0* beats *SOLVER 1.0 + P* by a large margin. It seems that in such early stages, there are not enough miaipairs. Thus the open boundary solver is more useful. By *End - 50*, *SOLVER 1.0 + P* has caught up to *SOLVER 2.0*'s performance. The combined *SOLVER 3.0* proves 27% more safe points than *SOLVER 2.0*. In *End - 25* and *End* stages, the improvements of *SOLVER 3.0* are 15% and 8% respectively.

5 Conclusions and Future Work

In this paper we have presented a partial-region safety-proving technique. From the experimental results presented above we may concluded that *SAFETY SOLVER 3.0* enhanced with this technique significantly outperforms previous solvers.

Below we provide two promising ideas for further enhancements.

1. Generalize region splitting techniques. The current technique is limited to adjacent miaipairs. (1a) An extension would be utilizing all possible single splitting points to divide a region. For example, in the left of Fig. 1, all four splitting points at $c1, e1, f1$ and $j1$ could be used in the safety search. (1b) A second extension would be to divide a region by other, larger gaps such as diagonal and one point jumps.
2. The aim of the current safety solver is to prove the safety status of territories. Applying the prover to real game playing and building a quick and strong heuristic safety analyzer for attacking or defending territory is a second interesting topic.

References

1. D.B. Benson. Life in the game of Go. Information Sciences, 10(2),17–29, 1976; Levy, D.N.L. (ed.): Reprinted in Computer Games, Vol. II, pp. 203-213. Springer, New York (1988)
2. Dyer, D.: An eye shape library for computer Go, `http://www.andromeda.com/people/ddyer/go/shape-library.html`
3. Müller, M.: Playing it safe: Recognizing secure territories in computer Go by using static rules and search. In: Matsubara, H. (ed.) Game Programming Workshop in Japan 1997, Tokyo, Japan, pp. 80–86. Computer Shogi Association (1997)
4. Niu, X., Kishimoto, A., Müller, M.: Recognizing seki in computer Go. In: van den Herik, H.J., Hsu, S.-C., Hsu, T.-s., Donkers, H.H.L.M(J.) (eds.) CG 2005. LNCS, vol. 4250, pp. 88–103. Springer, Heidelberg (2006)
5. Niu, X., Müller, M.: An improved safety solver for computer Go. In: van den Herik, H.J., Björnsson, Y., Netanyahu, N.S. (eds.) CG 2004. LNCS, vol. 3846, pp. 97–112. Springer, Heidelberg (2006)
6. Niu, X., Müller, M.: An open boundary safety solver in computer Go. In: van den Herik, H.J., Ciancarini, P., Donkers, H.H.L.M(J.) (eds.) CG 2006. LNCS, vol. 4630, pp. 37–49. Springer, Heidelberg (2007)

7. Vilà, R., Cazenave, T.: When one eye is sufficient: a static classification. In: van den Herik, H.J., Iida, H., Heinz, E.A. (eds.) Advances in Computer Games 10, pp. 109–124. Kluwer, Dordrecht (2003)
8. van der Werf, E.C.D.: AI techniques for the game of Go. PhD thesis, Maastricht University (2005)
9. Wolf, T., Pratola, M.: A library of eyes in Go, II: Monolithic eyes, In: Games of No Chance 3 (to appear, 2006)